**AIAA 93-4508**

# A Computational Architecture for Semi-autonomous Robotic Vehicles

T. W. Fong
NASA Ames Research Center
Moffett Field, CA

5. Brooks, R.A., "A Robust Layered Control System for a Mobile Robot", *IEEE Journal of Robotics and Automation*, 1986.

6. Albus, J.S., McCain, H.G., and Lumia, R., "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)", National Institute of Standards and Technology (NIST) Technical Note 1235, NIST, Gaithersburg, MD, July 1987.

7. Steele, R., Long, M., and Backes, P., "System Architecture for Asynchronous Multi-Processor Robotic Control System", AIAA 93-1159, AIAA/AHS/ASEE Aerospace Design Conference, Irvine, CA, February 1993.

8. Butler, P.L., and Jones, J.P., "A Modular Control Architecture for Real-Time Synchronous and Asynchronous Systems", *Proceedings of SPIE Applications of Artificial Intelligence 1993: Machine Vision and Robotics*, Orlando, FL, April 14-16, 1993.

9. Davis, G.J., "Computational Needs Survey of NASA Automation and Robotics Missions", NASA Technical Memorandum 103860, May 1991

10. Pittman, B., "Dynamic System Engineering: A Guide to Understanding and Implementing System Engineering", Pittman & Associates, 1991.

11. Ullman, M.A., "Experiments in Autonomous Navigation and Control of Multi-Manipulator, Free-flying Space Robots", Ph.D. thesis, Stanford University, Department of Aeronautics and Astronautics, Stanford, CA, March 1993.

13. Garvey, J.M., "A Russian-American Planetary Rover Initiative", AIAA 93-4088, AIAA Space Programs and Technologies Conference and Exhibit, Huntsville, AL, September 1993.

12. Lin, L., Simmons, R., and Fedor, C., "Experience with a Task Control Architecture for Mobile Robots", CMU-RI-TR 89-29, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 1989.

14. Schneider, S.A., Ullman, M.A., and Chen, V.W., "ControlShell: A Real-Time Software Framework", Real-Time Innovations, Inc., Sunnyvale, CA, 1992.

15. "Management Plan for the Human Exploration Demonstration Project", NASA Ames Research Center, Moffett Field, CA, December 1991.

16. Stoker, C.R., "Telepresence in the Human Exploration of Mars: Field Studies in Analog Environment", *Proceedings of Vision 21: Interdisciplinary Science and Engineering in the Era of Cyberspace*, Cleveland, OH, March 30-31, 1993

The development of TROV was performed during the summer of 1993. Since the vehicle uses the same ARCA subsystems as the *Mobile Exploration Landrover*, much of the TROV hardware and software systems were derived directly from that vehicle. Additionally, development of the vehicle controller was significantly enhanced by the use of commercial technology (i.e., VME processing hardware and VxWorks). In addition, the framework provided by ARCA greatly shortened development time by successfully enabling efficient teamwork and parallel subsystems development.

During the fall of 1993, TROV will be deployed in McMurdo Sound, Antarctica and controlled from multiple operator stations located at NASA Ames. The vehicle will be used in conjuction with VEVI and asynchronous processing (i.e., video frame capture, stereo image range correlation, etc.) to demonstrate the capabilities for performing science from a remote site. In this configuration, the complete system will demonstrate full utilization of the ARCA structure as shown in Table 6.

## V. Conclusion

ARCA provides a framework which enables the efficient team-based development of space telerobotic systems. The architecture was developed from a systems engineering perspective and is embodied by a core philosophy which supports extensible modular design and implementation. The design principles which guided the development of ARCA are reflected in the systems constructed with the architecture. Most importantly, it has been shown by these systems that modularity and the use of commercial technology can provide significant development benefits and increased system performance.

The architecture's capabilities have been clearly demonstrated by several operational telerobotic systems in field environments. These systems are characterized by short development time, modularity, flexibility, and a broad spectrum of computational needs. The success of these systems outside laboratory settings shows that ARCA provides the necessary tools and environment in which complex robotic systems can be rapidly and effectively constructed.

It must be noted, however, that the design of any robotic computational architecture does not end with the initial implementation. In order to maintain and further its success, ARCA is continuing to evolve to meet the needs of future space telerobotic systems. Systems engineering is always an interative process, hence, only through continued research and refinement can truly successful design be achieved.

Table 6. TROV and VEVI usage of ARCA

| Subsystem | Description |
|---|---|
| Processing hardware | embedded (VME bus components)<br>standalone (SGI/Sun workstations) |
| Standardized communications | Task Control Architecture |
| Synchronous processing | TROV Controller (VxWorks) |
| Loosely-synchronous processing | VEVI (WorldToolKit) |
| Asynchronous processing | Stereo range correlator<br>Video frame capture and storage |

## VI. Acknowledgements

The author gratefully acknowledges the contributions of Butler Hine III, Michael Sims, and Jay Steele to this work. The author would like to thank Chris Fedor and Reid Simmons, both of Carnegie Mellon University, for developing TCA and for their assistance. Lastly, the author would also like to thank Patrice Gelband of the Sense8 Corporation for her support of WTK.

## VII. References

1. Fong, T.W., Hine, B.P., and Sims, M.H., "Intelligent Mechanisms Group: Research Summary", NASA Ames Intelligent Mechanisms Group (IMG) internal document, Moffett Field, CA, January 1992.

2. Lavery, D. and Weisbin, C. "Telerobotics Program Plan", NASA Office of Advanced Concepts and Technology, January 1993.

3. Jones, J.P., Bangs, A.L., and Butler, P.L., "A System for Simulating Shared Memory in Heterogeneous Distributed-Memory Networks with Specializations for Robotic Applications", *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, France, May 12-14, 1992.

4. Man, R.F., "Subsumption architecture-based real-time multitasking kernel for programming autonomous robots", SPIE Conference on Cooperative Intelligence in Space III, November 1992

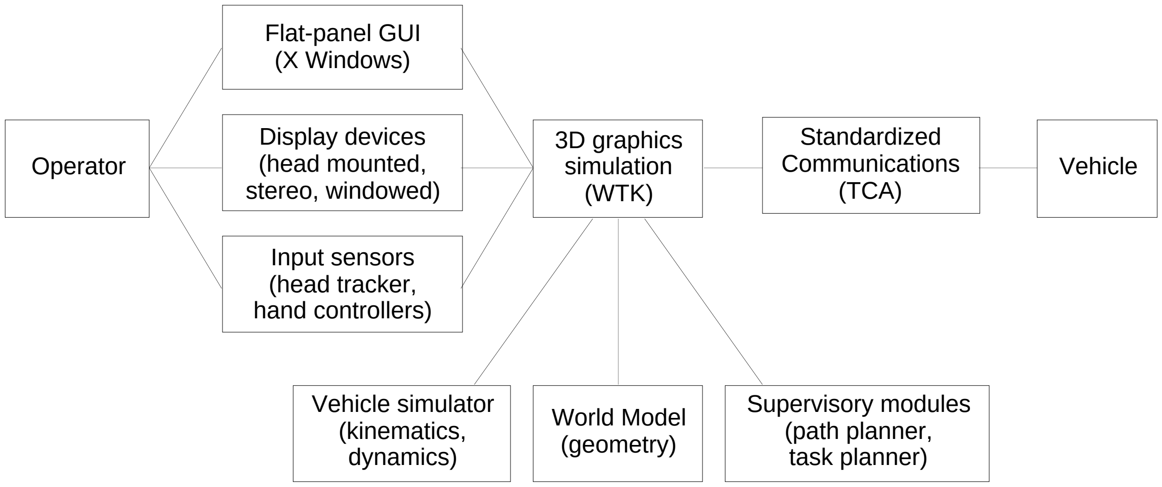

Flat-panel GUI (X Windows)
Display devices (head mounted, stereo, windowed)
Input sensors (head tracker, hand controllers)
Operator
3D graphics simulation (WTK)
Standardized Communications (TCA)
Vehicle
Vehicle simulator (kinematics, dynamics)
World Model (geometry)
Supervisory modules (path planner, task planner)

Figure 4: VEVI structure

by an on-board controller which performs tasks such as motion (rate) control and state estimation. Three ARCA subsystems are used: embedded processing hardware (VME bus single-board computer and peripherals), standardized communications (TCA), and synchronous processing (VxWorks).

The vehicle development has been, and continues to be, highlighted by the use of modular components. The on-board processing hardware, for example, is identical to other IMG robotic systems and enables the reuse of components (e.g., interface boards, sensors, etc.) from other devices. Additionally, the use of a common synchronous processing system (i.e., VxWorks) encourages consistent, modular software design and the sharing of code (e.g., control software, device drivers) with other real-time systems. More importantly, modularity has provided the significant benefit of increased systems reliability and robustness.

The most significant benefit realized with ARCA, however, is the flexibility provided by standardized communications. The use of TCA has allowed the *Mobile Exploration Landrover* to be quickly integrated with a variety of operator software including a command line interface, a X Windows based GUI, and VEVI. This has enabled teleoperation by a wide range of operators with a broad spectum of control capabilities. For example, the vehicle has been driven both from a home computer (i.e., an IBM PC) as well as VEVI running on a high-end Silicon Graphics workstation[13].

Telepresence Remotely Operated Vehicle

The *Telepresence Remotely Operated Vehicle* (TROV) is an underwater robotic vehicle designed for performing scientific field work in undersea environments[16]. The vehicle is being utilized as part of a joint pilot program between NASA and the National Science Foundation. The primary objective of this program is to use the Antarctic as an analog environment for the development and testing of systems for use in future space exploration.

TROV is a commercially available underwater *SuperPhantom II* vehicle (see Figure 3) built by Deep Ocean Engineering (San Leandro, CA) which has been extensively modified by the IMG for long-distance teleoperations. At present, the vehicle is equipped with four cameras (including pan/tilt stereo pair), acoustic navigation, several science sensors, and a simple manipulator arm. Overall system management is performed by an off-board controller which handles navigation, motion control (rate and station keeping), manipulator operation, sensor monitoring, and housekeeping tasks (e.g., lighting, camera functions). The controller, which uses the same components as other IMG systems, is connected to the vehicle by a 1,000 foot power and control umbilical.

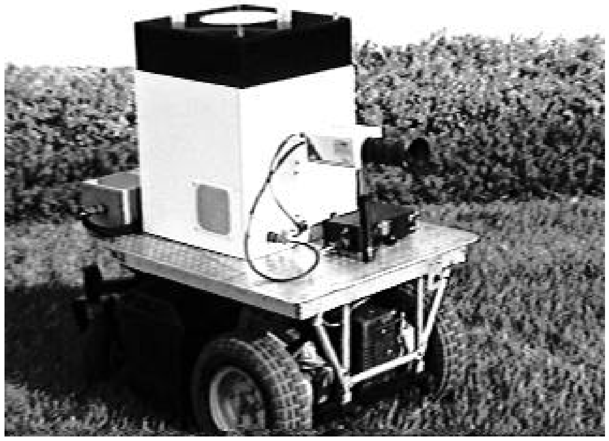
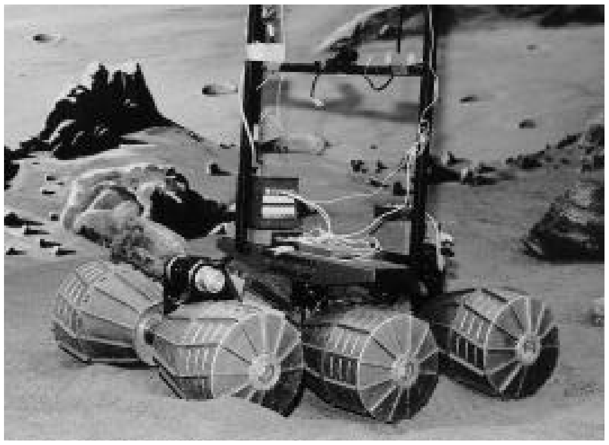
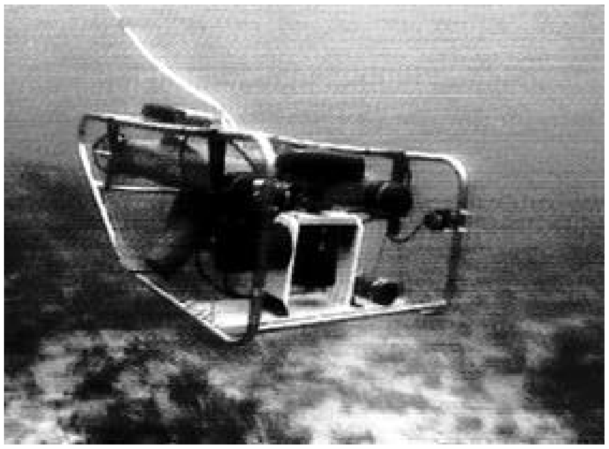


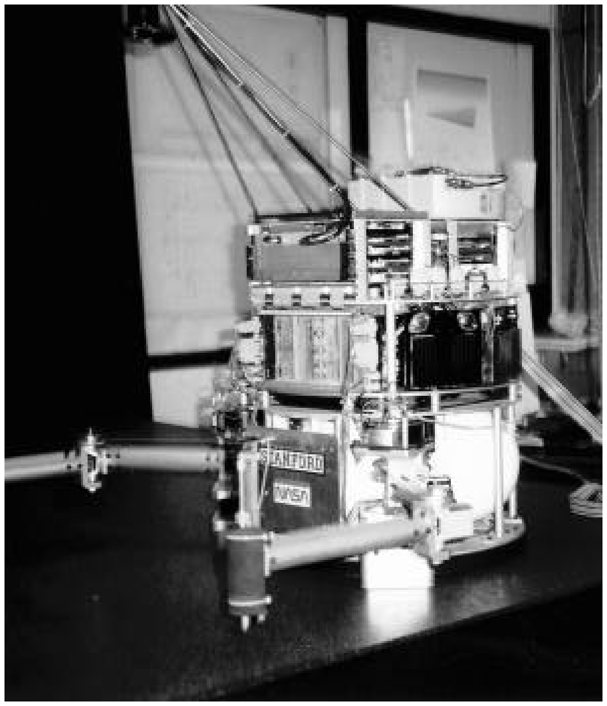
Figure 5: Mobile Exploration Landrover



(a)



(c)



(b)

Figure 3: Vehicles controlled with VEVI
(a) Russian Marsokhod rover (Moscow, May 1993)
(b) air-bearing floater (Stanford University, May 1993)
(c) Telepresence Remotely Operated Vehicle
(NASA Ames, July 1993)

controlled vehicle and the environment in which the vehicle is operating. Feedback from on-board vehicle sensors is used to update the simulated vehicle state and world model. Operators interact asynchronously with VEVI to control the graphical vehicle or to change inteface parameters (e.g., field-of-view, viewpoint). Under direct teleoperation, changes to the graphical vehicle are communicated in real-time via TCA to the actual device. For supervisory control, task-level command sequences are planned within VEVI then relayed to the vehicle controller for autonomous execution. This paradigm enables vehicle control in the presence of lengthy transmission delays and latencies, while increasing operator productivity and reducing fatigue. Figure 4 presents the overall structure of the VEVI system.

The ARCA subsystems included in VEVI are standalone processing hardware (Silicon Graphics workstations), standardized communications (TCA), and loosely-synchronous processing (WorldToolKit and X Windows). These elements have provided and continue to offer three important benefits. First, the use of commercially available technology made rapid software development possible. From conception to initial use required only two man-months effort. Second, modularity provides the flexibility required for fast integration of robotic devices into the VEVI system. Integration of the Russian Marsokhod rover, for example, was performed in only three days during February 1993[13]. Finally, standardized communications allows efficient long-distance operations. Though the Marsokhod integration time was short, the use of TCA enabled remote operations to be conducted in Moscow from NASA Ames (Moffett Field, CA)[13].

## Mobile Exploration Landrover

The *Mobile Exploration Landrover* is a semi-autonomous wheeled vehicle intended to support the *Human Exploration Demonstration Project* (HEDP)[15] and has been under development at NASA Ames since July 1992. The vehicle has two independent drive wheels and a variety of sensing devices (e.g., differential GPS, NTSC color cameras) as shown in Figure 5. Overall vehicle management is handled

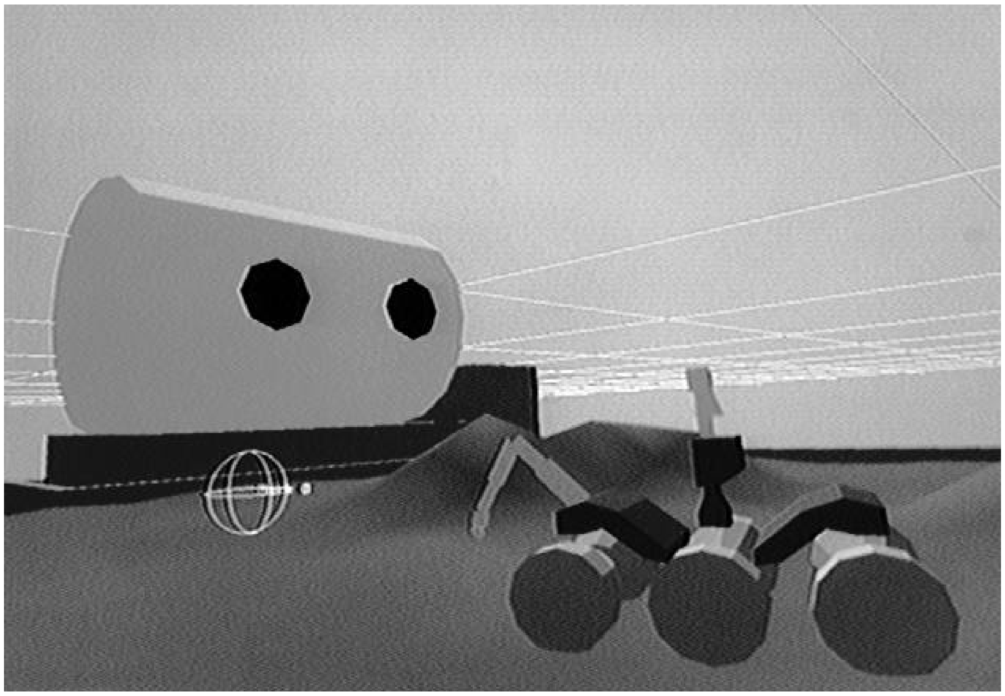


Figure 2: The Virtual Environment Vehicle Interface (VEVI)

graphical user interfaces (GUI) the *X Windows*™ and *Motif*™ protocols are utilized. These protocols provide a network transparent, portable system for client-server text and graphics applications. Windowed status displays (e.g., vehicle position strip chart) are easily built using GUI construction tools. Second, to support interactive 3-D graphical interfaces, the Sense8 Corporation's *WorldToolKit*™ (WTK) library is used. WTK provides an easy to use, platform independent environment for rapid prototyping and development of 3-D interfaces. This library enables the creation of stereoscopic virtual environment systems. Both X Windows/Motif and WTK applications are designed for C language development and can be executed on the UNIX workstations described previously. Additionally, X Windows/Motif has become a recognized standard and is expected to remain so.

### Asynchronous

ARCA's asynchronous processing includes those computational systems which utilize event (interrupt) driven or free-running processing without explicit program synchronization. Almost all non real-time and UNIX applications can be classified into the asynchronous domain. Representative applications include exception monitors, task level (strategic) planners, trajectory generators, vision systems, and decision making systems. These applications are modeless and typically are not synchronized to other modules.

Within ARCA, asynchronous processing may be performed on any type of processing hardware. In embedded systems, interrupt driven modules may be used to signal error conditions or to initiate reactive behavior. In standalone hardware, free-running vision systems may asynchronously generate range maps. No specific software restriction is given within this domain except that compatiblity with TCA is required for communications.

## IV. Case Studies

The following sections present case studies of operational robotic systems developed using ARCA. These systems demonstrate the flexibility of the computational architecture and the benefits realized during development of each.

### Virtual Environment Vehicle Interface

The *Virtual Environment Vehicle Interface* (VEVI) is a modular operator interface for direct teleoperation and supervisory control of robotic vehicles. VEVI utilizes real-time interactive 3D graphics and position/orientation sensors to produce a range of interface modalities from flat-panel (windowed or stereoscopic) screen displays to head-mounted/head-tracking stereo displays. A representative VEVI display is shown in Figure 2. The interface provides generic vehicle control capability and has been used to control wheeled, air bearing, and underwater vehicles in a variety of environments (see Figure 3).

VEVI executes as a loosely-synchronous process on Silicon Graphics workstations, rendering a scene with models of the

Table 4. ARCA embedded processing hardware (VME bus based)

| Type | Representative Use | Representative Model |
|---|---|---|
| Single-board computer | Uniprocessor with multitasking | Heurikon HK68/V3D (Motorola 68030 CPU) |
| Interface boards | Communications & device interfacing via standard and custom protocols | Xycom XVME-400 (RS232C)<br>Matrix MD-DAADIO (A/D, D/A, PIO)<br>National GPIB-1014 (IEEE-488) |
| Specialized boards | Optimized processing for specific embedded tasks | Matrox VIP 640 (NTSC frame capture)<br>PMC DCX-VM100 (Motor controller) |

Table 5. ARCA standalone processing hardware (UNIX workstations)

| Type | Representative Use | Representative Model |
|---|---|---|
| Sun Microsystems, Inc. | UNIX applications<br>Compute server<br>X Windows interfaces | SparcStation 4/370 (SPARC RISC CPU) |
| Silicon Graphics, Inc. | UNIX applications<br>Compute server<br>Interactive 3-D graphics | 4D/440 VGXT (MIPS R3000 CPU) |

The advantages of TCA, however, greatly outweigh its limitations. In particular, TCA provides a flexible, reliable, and easy to use communications method which greatly speeds program development. Centralized message routing, for example, facilitates the debugging of intermodule communications and coordination. Additionally, since TCA utilizes TCP/IP, it is intrinsically capable of long distance communications via the InterNet. The IMG, for example, has successfully used TCA to provide reliable communications between sites in California and France[13]. Lastly, the impact of centralized routing and transmission bandwidth can be reduced via appropriate usage, such as limiting communications to coarse-grained message passing.

Synchronous Processing

ARCA's synchronous processing encompasses all software systems which require a common clock, regular execution, strict execution schedule, or tight synchronization. This type of processing is colloquially known as *hard real-time,* and is typified by sampled data and control systems. To fulfill the requirements of this real-time domain, it is necessary to utilize an operating system which provides features such as multitasking, preemptive scheduling, low latency context switching, fast intertask synchronization, and bounded worst-case performance. This last feature is especially critical for guaranteeing a regular, strict execution schedule and meeting hard deadlines.

In ARCA, all synchronous processing systems is performed on embedded hardware using WindRiverSystem, Inc.'s *VxWorks*™ real-time operating system. VxWorks was chosen because it provides a well-integrated development and execution environment, an extremely efficient real-time kernel, standard networking facilities, and a high degree of UNIX compatibility. Thus, it is simple to integrate VxWorks applications with UNIX or networked based systems. In addition, VxWorks supports object-oriented toolsets, such as ControlShell, which facilitate robotic control software development[14].

Loosely-synchronous

ARCA's loosely-synchronous processing is similar to synchronous processing with one notable exception. Whereas synchronous processes requires strict execution and tight synchronization, loosely-synchronous ones do not. A virtual environment (VE) interface is representative of this domain. Although such interfaces may execute at real-time rates (e.g., 60 Hz), the vast majority do not require strict execution deadlines. Execution latencies or lag is not catastrophic as it would be for hard real-time processes. As such, a real-time operating system not an explicit requirement, though some benefit might be gained if one is utilized. Loosely-synchronous systems, however, do require some level of execution scheduling and coordination. The VE interface, for example, needs regular execution latencies to prevent users from experiencing disorientation.

Since the primary use of loosely-synchronous processing in ARCA are operator interfaces, two interface development environments were chosen. First, to support traditional

Table 3. ARCA subsystems

| Subsystem | Description |
|---|---|
| Processing hardware | Computational hardware including single-board computers, workstations, & multiprocessors |
| Standardized communications | Interprocess communications across a heterogeneous, distributed processing system |
| Synchronous processing | Computation requiring a common clock, regular execution, strict execution schedule, or tight synchronization |
| Loosely-synchronous processing | Computation similar to synchronous processing but without strict execution schedule or tight synchronization |
| Asynchronous processing | Computation which utilize event driven or free-running processing without synchronization |

IMG objective is to demonstrate potential space telerobotic scenarios via earth-based simulations. If actual flight systems were to be constructed, however, the modularity that ARCA provides would easily allow the replacement of unsuitable components with flight qualified ones.

Processing hardware

ARCA's processing hardware is heterogenous, distributed, and encompasses components required to support tasks ranging from real-time vehicle control to operator interfaces. The selection of a specific suite of hardware components is difficult due to the continual improvement of commercially available systems in conjuction with the on-going obsolescence of existing hardware. Thus, it is essential to select processing hardware which is not only appropriate to current needs, but also which provides an upgrade path for satisfying future requirements. The two types of ARCA processing hardware can be categorized as embedded and standalone.

The current embedded hardware is VME bus based and includes single-board computers, interface boards, and specialized processors (see Table 4). These components are used primarily for on-board vehicle processing and for synchronous tasks. VME bus was selected since it is a well established standard, has a large installed user and manufacturing base, and has a clear evolutionary path to future hardware. In addition, many VME manufacturers produce ruggedized and low power boards, which are advantageous for on-board vehicle systems.

Standalone processing hardware is currently a mixture of UNIX workstations as shown in Table 5. These systems are used primarily for non real-time processing, compute intensive applications, and operator interfaces. UNIX workstations provide tools, networking capability, and an environment well suited for software development. As with the embedded hardware, the systems shown in Table 5 were chosen due to large installed base and expected longevity. Silicon Graphics workstations, in particular, have shown significant improvements in recent years and provide unparalled real-time, interactive graphics capabilities.

The flexibility offered by the above hardware provides the extremely important benefit of commonality. All IMG robotic devices, for example, are able to utilize a common, modular set of embedded processing hardware. This directly enables reuse of hardware and software. Consequently, the effort required for development and maintenance is significantly reduced. Commonality, therefore, provides a method for greatly improving the efficiency and productivity of design.

Standardized communications

ARCA's standardized communications provides a system for interprocess communications and synchronization across a heterogenous and distributed processing system. Presenting a common programming interface, standardized communications facilitates teamwork by enabling modular development and reducing the difficulty of systems integration. At this time, standardized communications is achieved via the base layer of Carnegie Mellon University's *Task Control Architecture* (TCA).

TCA is a distributed, layered architecture with centralized control, communications occurs via coarse-grained message passing between modules[12]. The base layer of TCA implements a simple *remote procedure call* (RPC), in which the central control determines which module handles a particular message and in what order. This RPC interface operates via Berkeley Unix™ TCP/IP protocols and ethernet network transmission devices.

TCA was chosen because it is capable of providing interprocess communications between processes in all three ARCA processing domains and across a wide range of computing hardware. The primary limitation of TCA is that all communications are routed through the central control facility. As a result, the central process may become a bottleneck and communcations will deadlock if the process execution stops. Of secondary concern is the bandwidth provided by TCP/IP. TCA's effective throughput has been estimated to be approximately 200 kilobytes per second[12].

Table 2: Design principles of the Ames Robotic Computational Architecture (ARCA)

| Design Principle | Benefits |
|---|---|
| Modularity | • provides flexibility & extensibility<br>• eases design through encapsulation of complexity<br>• increases maintainability & reusability |
| Simplicity | • increases reliability<br>• speeds development & integration<br>• reduces learning time |
| Flexibility | • reduces brittleness from over-specification<br>• emphasizes importance of application specific needs<br>• encourages insertion of new technology |
| Heterogeneous Systems | • diversity increases optimization of subsystems<br>• enables incorporation of wide range of computational systems<br>• allows integration of diverse components |
| Commercial Technology | • use of "off-the-shelf" systems reduces development time<br>• reduces dependency on "one-of-a-kind" systems<br>• continual improvement due to market forces |

The fundamental philosophy of ARCA, however, is *to provide the minimal framework necessary to enable efficient team-based systems development of space telerobotic systems*. Since it is infeasible to ascertain *a priori* what needs all target applications will have, implementation details are not rigidly specified. In other words, ARCA does not stress a strict hierarchy or problem decomposition structure. Rather, the architecture offers development tools and an environment which encourages rapid, modular development and allows maximum flexibility to the developer. At the same time, ARCA provides a methodology for defining module interfaces and facilities for standardized communications. This allows design to proceed efficiently while maintaining modularity, flexibility, and extensibility of the system as a whole. As a natural consequence, concurrent teamwork is encouraged, complexity management is achieved, and the difficulty of systems integration is reduced.

## III. Implementation

### Overview

From a systems perspective, the ARCA framework is seen as the structure shown in Figure 1. There are five *computational* subsystems within the ARCA framework. The base subsystems are processing hardware and standardized communications. The other three systems are defined as processing domains: synchronous, loosely-synchronous, and asynchronous. Each processing domain utilizes elements of the processing hardware subsystem. Interaction between domains is handled via standardized communications. The subsystems are described in Table 3.

These subsystem definitions are fairly broad due to the difficulty of explicitly categorizing some types of processing. Control systems, for example, are primarily synchronous, yet may exhibit asynchronous behavior (i.e., halt on exception). Similarly, graphical user interface interaction is largely asynchronous, but may contain loosely-synchronous status updates.

The following sections provide summaries of each ARCA computational subsystem and presents details regarding current hardware and software components. It should be noted that many of these components are not suitable for flight systems. The processing hardware described below, for example, is not space-rated and is not likely to be in the near term. This is not considered a defect, however, since the

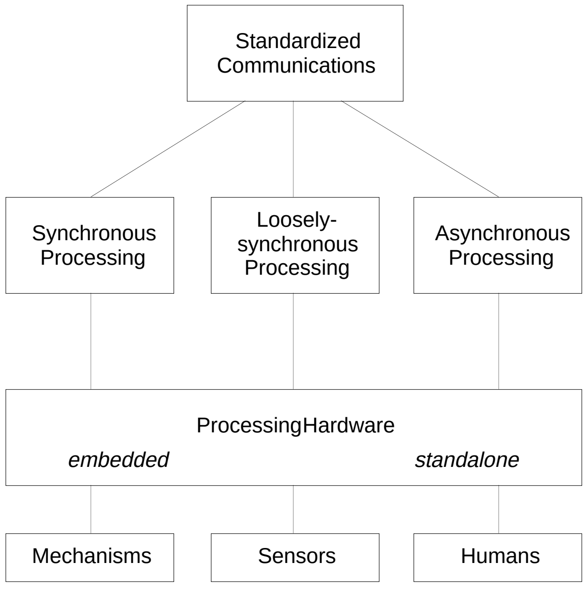


Figure 1: Ames Robotic Computational Architecture (ARCA)

Table 1. Rover computational processing requirements by scenario[1]

| Vehicle (speed) | Lunar survey rover (0.1 m/s) | Mars exploration rover (0.1 m/s) | Mars construction rover (1 m/s) | Lunar construction rover (10 m/s) |
|---|---|---|---|---|
| Capability/function | human operator plans, sends commands | Semi-autonomous navigation | Fully autonomous | Fully autonomous |
| Onboard navigation requirement | 0.5 to 2.0 MIPS | 1 to 10 MIPS | 500 to 5000 MIPS | 50 to 500 GIPS |
| Average travel time per cycle | 50 sec | 1.66 min | continuous | continuous |
| Average planning time per cycle | 10 sec | 12.7 min | continuous | continuous |

[1]from "Computational Needs Survey of NASA Automation and Robotics Missions"

Processing requirements for space telerobotic systems also vary significantly. Thus, the computational architecture must provide appropriate facilities and resources to match application specific processing needs. In the case of a planetary rover, for example, on-board navigation activities may involve manipulation and storage of large databases, stereometric range-finding & correlation, path planning and real-time command and control[9]. Estimates of these requirements depend on mission scenario and are shown above in Table 1.

It should be noted, however, that mission computing requirements are often ill-defined or may not be accurately specified. The figures given in Table 1, for example, are only coarse estimates of rover mission needs. Actual mission requirements, in fact, may vary by an order of magnitude and may not be known until late in the design phase. Thus, flexibility is also an important and necessary architectural element. Flexibility during both design and implementation allows better use of available technology, increases the utilization of newly available technology and provides a path for incorporating late design changes.

## Approach

The Ames Robotic Computational Architecture (ARCA) has been developed to fulfill these requirements and to provide a foundation for developing space telerobotic systems. The architecture specifically supports robotic operations utilizing direct teleoperation to time-delayed supervisory control in diverse environments. Thus, the architecture provides facilities for a wide spectrum of computational needs ranging from graphical operator interfaces to decision-making systems to on-board servo control. Furthermore, the architecture is intended to meet humans needs as well as autonomy. An intrinsic characteristic, therefore, is support for both asynchronous and synchronous processing. Finally, ARCA enables the timely development of maintainable, responsive, robust, and reliable devices. Central to this is a core framework and philosophy which supports extensible modular design and implementation.

ARCA was developed with a systems engineering philosophy. In this context, systems engineering is the iterative process of developing architectures, requirements, and interfaces in an iterative, hierarchical fashion to ensure that the system provides success through the balancing of performance, cost, schedule, and risk[10]. Effective use of systems engineering requires understanding the interrelationship between components through all project phases: concept formulation, design analysis, integration, and testing. Systems engineering, however, should not be viewed merely as a discipline, but rather as a basic philosophy from which to approach every aspect of design. In particular, it is important to view architectural design decisions and implementation in the context of a mission[11]. Thus, throughout the design of ARCA, emphasis was placed on creating a framework which enables the efficient development of *space telerobotic systems*, rather for generalized robotic devices.

ARCA's design incorporates the five guiding principles shown in Table 2. Modularity, simplicity, and flexibility are key elements for achieving clean, reliable, and robust system design. Modularity is of particular importance since it encourages the encapsulation of complex subsystems and independent testing before integration. The use of heterogeneous systems allows for subsystem optimization through the use of appropriate computational resources (e.g., graphics hardware for user interfaces, multiprocessing for compute-bound tasks). Finally, commercial technlogy provides significant benefits such as continual, market driven improvements in features, quality, and price/performance.

artificial intelligence (e.g., goal decomposition, hierarchical planning, blackboards) into a single framework for telerobotic control. The NASREM architecture defined a set of standard modules and interfaces intended to facilitate software design, development and validation[6]. The architecture is hierarchically layered and horizontally decomposed. Command data flow is strictly hierarchical, with high level commands decomposed spatially and temporally into subcommands.

One difficulty about the architecture is that it may be overly complex. Since NASREM attempts to codify the function and use of module interconnections, it tends to be restrictive and brittle when facing unforseen events. Additionally, the architecture design stems from a base philosophy that automation and reasoning systems can be explicitly decomposed and described. As a result, it is difficult to incorporate reactiveness and robustness into systems. Finally, different implementations of NASREM do not have common interface specifications which impedes integration.

### MOTES

In contrast to NASREM, a recent architecture developed for telerobotic control is the *Modular Telerobot Task Execution System* (MOTES). MOTES is the remote site software component of a telerobotic system which is being developed for space applications[7]. The primary goal of MOTES is to provide control and supervised autonomous control to support both space based operation and ground-remote control with time delay. In particular, it was designed for large physical & temporal separation between local and remote sites with potential for communication delays and varying latencies.

This architecture is designed to optimize task execution capability within a limited computational environment such as expected in flight systems. It, therefore, allocates the burden of processing on local (i.e., ground) sites which are expected to have far greater computational resources[7]. Thus, the architecture places task planning (asynchronous) responsibility at the local site and task execution (synchronous) responsibility at the remote site. The primary restriction of MOTES is the use of the Ada language. Consequently, execution and coordination is dependent solely on the Ada tasking model. Also, lack of methods for interfacing other languages places a severe burden on developers to create Ada based modules. This may prevent the use of certain hardware and software systems.

### MICA

Another recently developed robotic architecture is the *Modular Integrated Control Architecture* (MICA) developed at the Oak Ridge National Laboratory. It is a two-part software architecture which recognizes and exploits the differences between asynchronous and synchronous control[8]. The asynchronous portion simulates shared memory over a heterogeneous network which is used to communicate data and events for interprocess synchronization. The synchronous portion is based on a common clock within a block structure and utilizes interface processes for communication to other synchronous blocks and asynchronous processes.

Among the benefits MICA provides are extreme modularity, consistent programmatic interfaces, and precise synchronization for time-critical components. These factors have enabled large development teams to efficiently work on and complete projects. Specifically, MICA encourages modular construction and independent testing of subsystems before final assembly, which directly reduces integration problems and development time.

## III. Design

### Requirements

Space telerobotic systems have a myriad of requirements which a computational architecture must meet. The architecture must be capable of supporting operations ranging from direct teleoperation to time-delayed supervisory control. These operations can occur in varying environments; static to dynamic, structured to unstructured. At the same time, the architecture must reconcile the differing needs of human operators and robotic devices. Operator interfaces, for example, may require interactive real-time graphics while a path planner may require parallel processing hardware. In short, a computational architecture for space telerobotic systems must provide facilities for satisfying a broad spectrum of computational requirements and characteristics.

Control needs, in particular, differ widely in space telerobotic systems. At the servo level, synchronous, real-time operation is required to maintain controller stability, often with bandwidths up to a few kilohertz. Above the servo level, primitive task controllers (e.g., trajectory generators or exception monitors) may also operate synchronously but with reduced bandwidths in the tens of hertz. At higher levels, bandwidths may decrease further or control loops may become completely asynchronous. Also, the level of telerobotic automation, from direct operation to supervisory control, will directly impact overall system requirements. Direct master-slave telemanipulation, for example, is essentially synchronous and can require 100 Hz bandwidth with low (i.e., less than 0.5 sec) latency between master controller and slave device. Conversely, supervised teleoperation may be asynchronous and can tolerate significant time delays and latencies between operator and device.

nization. Thus, it is often unfeasible for a single person to manage the development and a team approach is required.

However, coordination of a development team is complex and may involve the integration of multiple researchers from different organizations, with different objectives, and with different levels of involvement and expertise[3]. In fact, this coordination problem is often the most difficult aspect of the development process. As a result, module definition, interface specification, complexity management, and systems integration become critical elements in robotic systems development. To achieve success, therefore, it is clear that a systems architecture which provides methods for efficiently handling these issues is needed.

From an engineering perspective, a well-defined architecture provides the underlying framework for achieving reliable, maintainable, cost-effective systems. Such an architecture does this by enabling methods for coherently integrating diverse physical, functional, and disciplinary subsystems. At the same time, the architecture promotes efficient design by facilitating the development of implementation specifics. For robotic devices, in particular, a well-defined *computational architecture* offers three important benefits. First, it provides a disciplined methodology for building complex systems from smaller components. Second, it offers an environment for coordinating processes, resolving conflicts, and handling exceptions. Most importantly, though, it provides a core framework for unifying disparate modules such as user interfaces, control systems, sensors, mechanisms and processors into a cohesive system.

## The Ames Robotic Computational Architecture

It is extremely important to recongize, however, that a single computational architecture is not appropriate for all tasks. In fact, almost all architectures will work well on simple laboratory cases, but perform poorly on real-world problems. Additionally, the choice of an architecture is strongly domain and task dependent. What works well for one application may fail for another. Moreover, there have been too few robust, working systems to generalize or conclude what an optimum architecture looks like. Or even what *optimum* means. Yet, it has been shown that the use of an appropriate architecture can greatly impact and enhance the life-cycle of a robotic system.

Given the task of efficiently constructing space telerobotic systems, therefore, the IMG has developed the Ames Robotic Computational Architecture (ARCA). ARCA provides a unified framework for integrating diverse subsystems and components, ranging from servocontrollers to operator interfaces. In addition, it reconciles a broad spectrum of processing domains, enables the development of complex systems and reduces the difficulty of systems integration. ARCA differs from other systems by its reliance on commercially available technology and by its core systems philosophy. The architecture was developed using a systems engineering approach with emphasis placed on understanding the interrelationship between system components and on maximizing the overall system performance.

## II. Related Work

There have been significant advances in robotic computational architectures during the past decade. System developers have typically relied upon these architectures to guide the construction of robotic devices and for providing computational services (e.g., communications, processing, etc.) to subsystems and components. These architectures, however, have tended to be task and domain specific and have lacked suitability to a broad range of applications. For example, an architecture well suited for direct teleoperation tends not to be amenable for supervisory control or for autonomous use.

One recent trend in robotic computational architectures has been a focus on behavior-based or reactive systems. Behavior based refers to the fact that these systems exhibit various behaviors, some of which are emergent[4]. These systems are characterized by tight coupling between sensors and actuators, minimal computation, and a task-achieving "behavior" problem decomposition[5]. Although such systems have shown promise and have favorable attributes (e.g., robustness, minimal processing requirements), the underlying architecture appears to be suitable only for a limited range of applications. In particular, behavior-based architectures work well when system scaleability, human-machine interaction or task complexity are not overriding concerrns.

The other leading architectural trend is typified by a mixture of asynchronous and synchronous control and data flow. Asynchronous processes are characterized as loosely coupled and event-driven without strict execution deadlines. Most graphical user interfaces operate asynchronously. Synchronous processes, in contrast, are tightly coupled, utilize a common clock and demand hard real-time execution. Servocontrollers and sampled data systems are representative synchronous processes. In general, systems built in this architectural style are larger, more complex and employ traditional (e.g., "top-down", "bottom-up") decompositions than strictly behavior based systems.

### NASREM

One of the earliest computational architectures is the *NASA/NBS Standard Reference Model for Telerobot Control* (NASREM). Developed during the early 1980's, NASREM was an early attempt to incorporate many concepts from

# A COMPUTATIONAL ARCHITECTURE FOR SEMI-AUTONOMOUS ROBOTIC VEHICLES

Terrence W. Fong[*]
NASA Ames Research Center
Moffett Field, California

## Abstract

This paper describes a computational architecture which supports the development and operation of semi-autonomous robotic systems. The Ames Robotic Computational Architecture (ARCA) provides a unified framework for integrating diverse subsystems and components, ranging from servocontrollers to operator interfaces. In addition, it reconciles a broad spectrum of processing domains, enables the development of complex systems and reduces the difficulty of systems integration. ARCA differs from other systems by its reliance on commercially available technology and by its core systems philosophy. The architecture was developed using a systems engineering approach with emphasis placed on understanding the interrelationship between system components and on maximizing the overall system performance. The requirements, design and implementation of the architecture are presented together with case studies of operational robotic systems.

## I. Introduction

### Background

The Intelligent Mechanisms Group (IMG) at the NASA Ames Research Center is an applied research team engaged in the development of space robotic systems. This application focus is driven by the importance of intelligent mechanisms technology to future NASA missions[1]. In particular, the IMG is investigating systems in support of the NASA Telerobotics Program. The primary goal of this program is to develop, integrate and demonstrate telerobotic technologies which will lead to increasing the operational capability, safety, cost effectiveness and probability of success of NASA missions[2]. Thus, the IMG seeks to develop systems to fulfill this objective and to advance human-machine capabilities through the implementation of space robotic systems.

The development of such systems, however, is a complex task which is constrained by several factors. First, the appropriateness of providing and utilizing autonomy must be determined for each application. Mission guidelines, for example, might dictate the use of suited astronauts for certain satellite servicing tasks, whereas automated construction might be broadly applicable to planetary habitat construction. Second, environmental factors and resource availability will directly impact any set of tasks. On-board processing for a planetary rover will be constrained by mass and power restrictions. Hence, capacity for automation may be severely limited. Finally, it is necessary to understand the consequences of mixing humans and robotic systems. This knowledge is crucial for unifying the two in such a way that the advantages of both are magnified and that the limitations of both are minimized[2].

Current IMG efforts are focused on developing space telerobotic systems which utilize a mixture of teleoperator and robotics technologies. These systems are semi-autonomous; they exhibit some on-board autonomy but are primarily intended to be controlled by supervising humans. They are targeted for applications (e.g., space construction, remote operations on planetary surfaces, habitat assembly) for which supervisory control can return significant benefits. Thus, a certain level of autonomy is necessary to guarantee safe, nominal system operation. Additionally, these space telerobotic systems differ from existing terrestrial counterparts. In contrast to current ground-based systems, for example, space robotic devices must provide support for a mixture of manual and automated control, operate in changing and unstructured environments, and handle variable communications delays and latencies betwen the operator and the remote system. Lastly, these systems are being developed to demonstrate potential mission scenarios through earth-based simulations. As such, they are intended to be operated and evaluated in field environments rather than laboratory settings.

### Need for a Computational Architecture

Robotic systems are complex and tend to be difficult to develop. They integrate multiple sensors with effectors, have many degrees of freedom and must reconcile hard real-time systems with systems which cannot meet real-time deadlines[3]. Multiple data streams (e.g., sensory, command, knowledge) from multiple sources must be processed into action in a timely fashion. Integration is difficult because it requires the assembly of many complex subsystems into a structure with high degrees of interconnection and synchro-

[*]Recom Technologies, Inc., Member, AIAA